KnBest - A Balanced Request Allocation Method for Distributed Information Systems

Jorge-Arnulfo Quiane-Ruiz, Philippe Lamarre, Patrick Valduriez

K_nBest - A Balanced Request Allocation Method for Distributed Information Systems*

Jorge-Arnulfo Quiané-Ruiz**, Philippe Lamarre, and Patrick Valduriez

INRIA and LINA
Université de Nantes
2 rue de la houssinière, 44322 Nantes Cedex 3, France
{Jorge.Quiane,Philippe.Lamarre}@univ-nantes.fr,Patrick.Valduriez@inria.fr

Abstract. In large-scale distributed information systems, providers are typically autonomous, i.e. free to leave the system at will or to perform certain requests. In this context, request allocation is critical for the efficient system's operation. However, most methods used in distributed information systems aim at maximizing overall system performance (throughput and response times) by allocating requests to the most efficient providers, without considering providers' autonomy. In this paper, we propose a balanced request allocation method, K_nBest, which considers providers' autonomy in addition to load balancing. Our method is general and simple, so that it can be easily incorporated in existing distributed information systems. We describe the implementation of K_nBest in different scenarios. Finally, we give an experimental evaluation which shows that K_nBest significantly outperforms traditional request allocation methods.

1 Introduction

We consider distributed information systems that are dynamic and provide access to a large number of information providers. Providers can be fairly autonomous, i.e. free to leave the system at will and to express their *intentions* for performing requests. Their *intentions* can stem from combining their *preferences* with other important factors such as their *strategies*.

In this context, allocating requests to providers must maximize overall system performance (throughput and response times). The traditional solution, used in mediator systems [6, 14, 17], is to reduce the *overloading* of providers as much as possible so that load balancing is increased. However, preserving the providers' *intentions* when performing request allocation is equally important to keep providers happy and the system stable. With the traditional solution, providers can become unsatisfied when their intentions are not met and simply quit, resulting in an unstable system. Therefore, request allocation should

* Work partially funded by ARA "Massive Data" of the French ministry of research (projects MDP2P and Respire) and the European Strep Grid4All project.
** This author is supported by the Mexican National Council for Science and Technology (CONACyT).

also take into account providers' *intentions*. This is a timely problem with the deployment of web services and service-oriented architectures.

Providers' *intentions* are dynamic and depend on the providers' context. For instance, some providers may desire to perform specific requests at some time, but not at another time. Furthermore, providers can be heterogeneous in terms of capacity and data. Heterogeneous capacity means that some providers are more powerful than others and can treat more requests per time unit. Data heterogeneity means that different providers provide different data and thus produce different results for the same request.

In this paper, we address the request allocation problem by considering providers' *intentions* in addition to load balancing. We propose a balanced request allocation method, called $K_n Best$. It is inspired by the *two random choices* paradigm which has proven useful for dynamically assigning tasks to providers [10, 20, 21]. Our method is general and simple, so that it can be easily incorporated in existing distributed information systems. It generalizes traditional methods and can be adapted to the application by varying several parameters. We describe the implementation of $K_n Best$ in different scenarios. Finally, we give an experimental evaluation which compares $K_n Best$ to traditional request allocation methods. We show that, with autonomous information providers in the system, our method significantly outperforms these methods

The rest of this paper is organized as follows. In Section 2, we define the system model and introduce some notations. In Section 3, we present the $K_n Best$ method. In Section 4, we give the experimental evaluation of $K_n Best$. In Section 5, we discuss related works. Section 6 concludes.

2 Model and Notations

We consider a system consisting of a set of mediators, M, of a set of consumers, C, and of a set of providers, P. These sets are not necessarily disjoint, i.e. we can have $M \cap C \cap P \neq \emptyset$. Requests are formulated in a format abstracted as a triple $q = < c, d, n >$ such that $q.c \in C$ is the identifier of the consumer that has issued the request, $q.d$ is the description of the task to be done, and $q.n \in \mathbb{N}^*$ is the number of providers to which the consumer wishes to allocate its request. Consumers send their requests to a mediator $m \in M$ which allocates each incoming request q to the $q.n$ providers in P_q, where P_q denotes the set of providers that can treat q. We use N_q for denoting $||P_q||$, or simply N for the sake of simplicity when there is no ambiguity on q. In the case where $q.n > N_q$, the consumer $q.c$ will get N_q results instead of $q.n$. We only consider the arrival of feasible requests, that is, those requests for which $P_q \neq \emptyset$.

Request allocation of some feasible request q among the providers which are able to deal with q is defined as a vector $\overrightarrow{All\bar{o}c}$ or $\overrightarrow{All\bar{o}c}_q$ when there is an ambiguity on q (see Definition 1). The set of providers such that $\overrightarrow{All\bar{o}c}[p] = 1$ is noted $\widehat{P_q}$.

Definition 1. ***Request Allocation***
The allocation of request q among the providers in P_q is a vector $\overrightarrow{All\bar{o}c}$ of length

N *such that:* $\forall p \in P_q$, $All\vec{oc}[p] = \begin{cases} 1 & \text{if provider } p \text{ gets the request} \\ 0 & \text{otherwise} \end{cases}$

with $\sum_{p \in P_q} All\vec{oc}[p] = min(q.n, N)$

Each provider $p \in P$ has a finite *capacity*, $cap(p) > 0$, for performing feasible requests. The *capacity* of a provider denotes the number of computational units that it can have. Similarly, each feasible request q has a *cost*, $cost_p(q) > 0$, that represents the computational units that q consumes at p. We define provider *utilization* as follows.

Definition 2. ***Provider Utilization***
Let Q_p denote the set of requests that have been allocated to p but have not already been treated at time t (i.e. the pending requests at p). The utilization of a given provider $p \in P$ at time t, $U_t(p)$, is defined as the computational units that Q_p consumes at p

$$U_t(p) = \frac{\sum_{q \in Q_p} cost_p(q)}{cap(p)}$$

Providers are free to express their ***intentions*** for performing each feasible request q, denoted by the $PI_p(q)$ function whose values are between $-\infty$ and 1, and where p denotes a given provider. If the *intention* value is positive, the greater it is, the greater the desire for performing requests. If the *intention* is negative, the smaller it is, the greater the refusal for performing requests. Providers' refusal can go down to $-\infty$ because *utilization* can, theoretically, grow up to $+\infty$. In order to guarantee system stability, the way in which such *intentions* are computed is considered as private information for providers.

3 K_nBest Method

In this section, we present the K_nBest method for balanced request allocation. It is meant to be run by one or several mediators which allocate the requests they receive from the clients. We restrict ourselves to the case where requests can be viewed as single units of work called *tasks*. An incoming feasible request q can be allocated to n providers (because of data heterogeneity, more than one provider can provide answers). We assume that there is a matchmaking mechanism ([8] for example) to find the set P_q of providers that are able to deal with each incoming feasible request q. Therefore, we can focus on request allocation only. Our method is inspired by the *two random choices* paradigm [10, 20, 21]. The principle of *two random choices* is to randomly select a set of providers K among the N_q providers and then allocate the request to the least utilized provider in K. K_nBest uses a similar principle. We describe below the principle and properties of the K_nBest.

3.1 K_nBest Principle

Given an incoming feasible request q and the set P_q of providers which are able to deal with it, we denote the providers' *intention* for dealing with q by the vector $\overrightarrow{PI}^q$. Algorithm 1 shows the main steps of K_nBest for allocating q.

First (line 2 of Algorithm 1), a set K of providers are selected at random among the set P_q, where $||K|| = k$ and $k \in \mathbb{N}^*$. Second (line 3), the $k_n \in \mathbb{N}^*$ least utilized providers (i.e. the K_n set) are selected among the set K of providers[1]. Third (line 4), the providers' *intention* vector, $\overrightarrow{PI}^q[p]$, for dealing with q is computed. Considering our system architecture [15], this operation is realized at the mediator sites and do not impact the network traffic. Next (line 5), the *intentions*' ranking vector $\overrightarrow{R}^q$ is computed. This ranking is necessary for the selection of providers to deal with q. Intuitively, $\overrightarrow{R}^q[1]$ is the most interested provider to deal with q, $\overrightarrow{R}^q[2]$ the second, and so on until to $\overrightarrow{R}^q[N]$ which is the least interested. Finally (lines 6-8), the request q is allocated to the n best providers. If $N < n$, the request is simply allocated to all N providers without any further considerations. The second (line 3) and fourth (line 5) phases can be solved using a sorting algorithm. So, in the worst case, their complexity is $O(k \log_2(k))$ and $O(k_n \log_2(k_n))$, respectively.

Algorithm 1: $K_nBest\_Providers$

Input : q, k, k_n, P_q
Output: $\overrightarrow{Alloc}_q$

1 **begin**
2 $K \leftarrow$ select, at random, k providers in the set P_q ;
3 $K_n \leftarrow$ select the k_n less utilized providers in the set K ;
4 **foreach** $p \in K_n$ **do fork** ask for the provider's *intention* $\overrightarrow{PI}^q[p]$;
5 compute the providers' *intention* vector ranking $\overrightarrow{R}^q$;
6 **for** $i = 1$ **to** $min(n, k_n)$ **do** $\overrightarrow{Alloc}_q[\overrightarrow{R}^q[i]] \leftarrow 1$;
7 **for** $j = min(n, k_n) + 1$ **to** k_n **do** $\overrightarrow{Alloc}_q[\overrightarrow{R}^q[j]] \leftarrow 0$;
8 **end**

3.2 K_nBest Property and Analysis

An expected property of providers is *individual rationality*, whereby they behave according to their own interests only [1]. However, this may lead providers to not participate in a given request allocation since they are *selfish* and may have no interest in processing the request. Thus, *individual rationality* is incompatible with the system's objectives to process efficiently all incoming feasible requests and satisfy consumers. For these reasons, we are looking for a special kind of *long-term individual rationality* which K_nBest allows.

[1] We can indifferently assume that the values of k and k_n are predefined by the administrator or defined on the fly by mediators.

In the case of m requests and m homogeneous providers, request allocation methods based on the *two random choices* paradigm (which we call the $TRCBased$ methods) assign to providers a maximum load of only $\theta(\log\log m)$ with high probability [10, 20]. Given an incoming request q, $TRCBased$ methods randomly select two providers (the K set) among P_q and allocate q to $p \in K$, such that

$$\forall\, p' \in K \backslash \widehat{P_q},\ U_t(p') \geq U_t(p)$$

Nonetheless, the $TRCBased$ methods are only suitable for homogeneous distributed systems (i.e. providers have the same *capacity* for performing requests). In the context of heterogeneous distributed systems, request allocation methods ensure good performances in the system by considering the capacities of all the providers in P_q [6, 12, 14, 17, 21]. These methods (the $CapacityBased$ methods) allocate each incoming request q to $p \in P_q$ such that

$$\nexists p' \in P_q \backslash \widehat{P_q} :\ U_t(p') < U_t(p)$$

A third possible way to allocate requests is by only considering the providers' *intentions*, which we call the $IntentionBased$ methods. Given the request q to be allocated, $IntentionBased$ methods allocate q to $p \in P_q$ such that

$$\nexists p' \in P_q \backslash \widehat{P_q} :\ PI_{p'}(q) \geq PI_p(q)$$

On the other hand, the k_n parameter defines the general behavior of the K_nBest method. For smaller values of k_n, K_nBest allocates the requests by mainly considering the providers' *utilization*, and by basically considering the providers' *intentions* for greater values of k_n. Thus, the value that k_n may take depends on the type of application. Therefore, we recommend smaller values of k_n if the system pays more attention to the providers' *utilization* and greater values if it pays more attention to the providers' *intention*. In the following theorem, we summarize the K_nBest properties which bound its behavior.

Theorem 1. *For a large number of heterogeneous providers, K_nBest behavior is bounded by the following properties.*

(i) if $k = 2q.n \wedge k_n = q.n$, K_nBest is equivalent to the $TRCBased$ methods.
(ii) if $k = N_q \wedge k_n = q.n$, K_nBest is equivalent to the $CapacityBased$ methods.
(iii) if $k = N_q \wedge k_n = k$, K_nBest is equivalent to the $IntentionBased$ methods.

Proof. Omitted because of triviality.

A different way to proceed in K_nBest is by selecting first the most *interested* providers (K_n) among K to finally allocate requests to the least *utilized* providers among K_n. If request allocation is done this way, *intentions* are given more importance than *utilization*. Besides, one can replace the providers' *intentions* by the consumers' *intentions* for allocating requests if the objective of the system is, for example, to ensure interesting results for consumers. If the system's goal is to satisfy both providers and consumers, the consumers' *intentions* can be considered as an additional step in K_nBest, or *intentions* of both providers and consumers can be merged in only one step.

4 Experimental Evaluation

In this section, we give an experimental evaluation of K_nBest using simulation. We carried out several series of tests with a main objective in mind: *how well K_nBest operates in environments with heterogeneous and autonomous providers.* We assume a single mediator. In order to assess the quality of K_nBest, we conducted three types of evaluations: *performance*, *request load balance*, and *satisfaction balance.* In the following, we introduce the baseline methods to which we compare K_nBest and our experimental setup, and then we present the experimental results.

4.1 Baseline Methods

In distributed information systems, the well-known methods to allocate requests efficiently across providers are those with the $CapacityBased$ behaviour (i.e. the $CapacityBased$ methods) [6, 14, 17]. These methods, unlike the $TRCBased$ ones, operate well in heterogeneous systems which is why we use them as baseline.

Economic models have been shown to provide efficient request allocation in heterogeneous systems [3, 4, 16, 13]. We use, as baseline, an *Economic* method whose criterion to select providers is the *utilization* and *bid* of providers. Note that different economic methods may have other performance results than those presented here. In our experiments, we use virtual money (*token*) which is just seen as a means of regulation. In the course of time, the *tokens* are spent by providers in order to acquire requests. Thus, a source of financing is necessary to them. Otherwise, after some time, providers would not have more *tokens* to bid positively. We have chosen to associate a bank with the mediator, which gives a specific amount of *tokens* to providers at the registration step and equally redistributes the *tokens* in the course of time.

4.2 Experiment Setup

We built a Java-based simulator that models a *mono-mediator* distributed system, following the system architecture in [15]. In all experiments, the number of consumers and providers is 200 and 400 respectively, with only one mediator allocating all the incoming feasible requests. We assign sufficient resources to the mediator so that it does not cause bottlenecks in the system. Each result we present is obtained by 10 simulation series. Feasible requests arrive to the system following the *Poisson* distribution, which has been found in dynamic and heterogeneous distributed systems [5].

For our simulations, we consider that providers provide answers with similar quality, hence assuming that consumers always ask for 1 provider to solve their requests (i.e. $q.n = 1$). Since we focus on heterogeneous distributed systems, we set $k = N_q$. Basing ourselves on the results of [10, 20], we set $k_n = 2q.n$ (i.e. $k_n = 2$). In order to compare the K_n providers, we consider their *intentions* for performing requests. Providers work out their *intentions* using the S_bQLB method [7]. To ensure high autonomy in our experiments, providers' *preferences*

Table 1. Simulation parameters.

Parameter	Definition	Value
nbConsumers	Number of consumers	200
nbProviders	Number of providers	400
nbMediators	Number of mediators	1
qDistribution	Query arrival distribution	Poisson
iniSatisfaction	Initial providers' satisfaction	0
qClasses	Supported query classes	2
nbSimulations	Number of realized simulations for each experience test	10

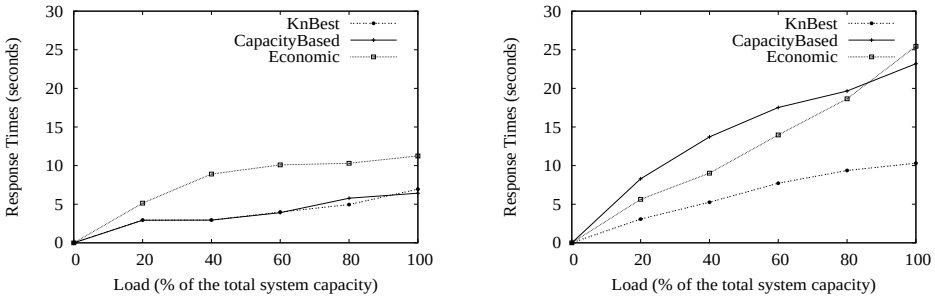
are randomly obtained between -1 and 1. More sophisticated mechanisms for obtaining such preferences can be applied ([2] for example).

Since our main focus in this paper is to study the way in which requests are allocated in the system, we do not take into account the bandwidth problem. It is because of this that simulation tests are enough for evaluating our method. Providers are initialized with a satisfaction value of 0. We set the providers' capacity heterogeneity in accordance to the results in [18]. Based on these results, we generate around 10% of low-capable, 60% of medium-capable, and 30% of highly-capable providers. The highly-capable providers are 3 times more capable than medium-capable providers and still 6 times more capable than low-capable ones. We generate two classes of requests that consume, respectively, 130 and 150 computational units at the high-capable providers (taking about 1.3 and 1.5 seconds, respectively).

4.3 Experimental Results

Performance. We start with an evaluation of the system's response time guaranteed by the *CapacityBased*, *Economic*, and K_nBest methods. As is conventional, we define the response time as the elapsed time from the time a consumer issues the request to the time it receives the answer. As first case, we consider that providers are *captive* and do not leave the system by *unsatisfaction*, *request starvation*, nor *overutilization*. Figure 1(a) shows that K_nBest yields the same response times as *CapacityBased*, and that both of them outperform the *Economic* method. This shows that K_nBest is also suitable for environments where providers do not leave the system.

As second case, in order to evaluate the impact of provider departures, we study response time when providers may leave the system by *unsatisfaction* or *request starvation*. To do so, we allow providers to leave the system if their *satisfaction* is 0.15 under their *adequation*. The *adequation* notion (see [7] for details) denotes the degree of satisfaction towards all the feasible requests proposed by the system. Also, providers are allowed to leave the system if they do not perform at least 25% of what they should. The results are shown in Figure 1(b). We observe that K_nBest significantly outperforms the *CapacityBased* and *Economic* methods. While K_nBest degrades in average the system's response time by a factor of 1.5 only, the *Economic* and *CapacityBased* do it, respectively,



(a) Providers do not leave the system

(b) Providers may leave the system

Fig. 1. Response Time.

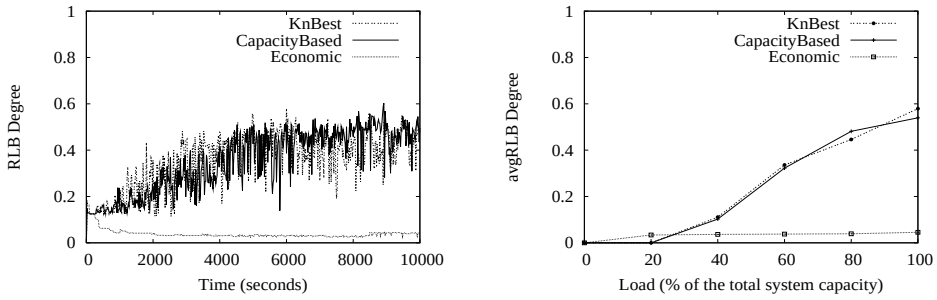
by a factor of 1.8 and 4! We observed that in the *Economic* method, providers usually quit by *request starvation*, while in the *CapacityBased*, they do so by *unsatisfaction*.

We ran other experiments where providers may also leave the system by *overutilization*[2], but by lack of space, we do not present the results here. We observed that $K_n Best$ and $CapacityBased$ do not suffer from providers' departures by *overutilization*, while the performance of the *Economic* method is degraded by a factor of 4.5 and, worse again, at a given time the system remains with no providers! These results demonstrate the great impact on response times of providers departures by *unsatisfaction*, *request starvation*, and *overutilization*. Therefore, in a system where providers are *selfish*, we must pay special attention to their interests.

Request Load Balance. We now study the *request load balance* (RLB) for various workloads. In the experiments, we measure the RLB at any time as the ratio of the smallest and largest *utilized* providers. Furthermore, it is important that the system strives to give requests to all providers in the system if possible so they don't leave. Thus, we also measure the *average request load balance* ($avgRLB$) at a discrete time interval $[t_1, t_2]$. The $avgRLB$ measure is symmetrical to the RLB measure. The *average utilization*, $U_{[t_1,t_2]}(p)$, is said to be the average *utilization* of the provider p at the discrete time interval $[t_1, t_2]$.

We know that the thresholds of *request starvation* and *overutilization* over which providers decide to leave, are very subjective and might depend on several external factors. Thus, to avoid any question on the choice of such *thresholds*, we assume, in this study, that providers are not allowed to leave the system whatever their degree of *request starvation* and *overutilization* are.

[2] For instance, when providers want to guarantee a good *quality of service*.



(a) For a workload from 30 to 100%

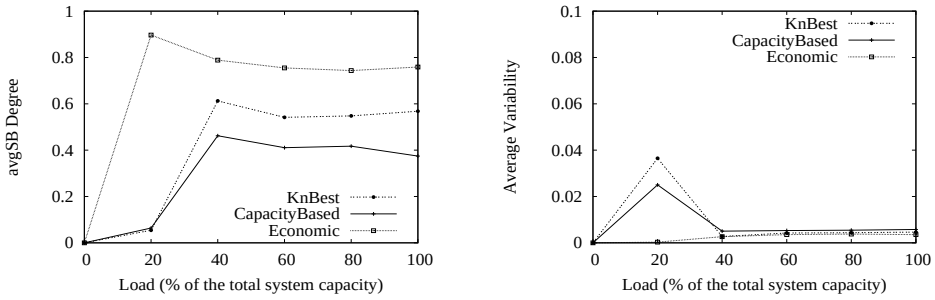
(b) In average for different workloads

Fig. 2. Request load balance.

Contrary to the expected, on the one hand, the results show that the *Economic* method has serious problems to ensure good *RLB* ratios in the system, despite that providers take more into account their *preferences* than their *utilization* at the time of bidding for requests. On the other hand, the results show that *CapacityBased* and K_nBest have problems to ensure good *RLB* only for workloads under the 40% of the total system capacity. When workloads increase, both algorithms improve the *RLB* in the system. This is because most providers in the first case have almost no requests (i.e. have all their capacity available), and thus, requests may be allocated to those providers that spend more treatment units to perform them. Hence, each time that requests are allocated to the least capable providers, the distance between the most and least utilized providers increases significantly. For lack of space, we do not present these results.

In order to explain the above phenomenon, we present the results of a series of simulations where we uniformly vary the workload from 30% (at the beginning of the simulation) to 100% (at the end of the simulation). The results show that *CapacityBased* and K_nBest methods effectively improve the *RLB* as the workload increases, and that the *Economic* method cannot ensure an acceptable *RLB* in the system (see Figure 2(a)). In all cases the K_nBest performance is as good as *CapacityBased* one, even if the former takes into account the providers' *intentions* and the latter does not.

Let us analyze the *avgRLB* guaranteed by these three methods, that is, how well these methods avoid *request starvation*. To this end, we measure the *avgRLB* in a time interval of 20 seconds over a simulation of 10000 seconds for different workloads. The results are shown in Figure 2(b). We observe that, unlike the *RLB* results, for workloads over 20% of the total system capacity the *CapacityBased* and K_nBest methods significantly outperform the *Economic* method. This means that while *CapacityBased* and K_nBest strive to allocate requests on giving requests to all the providers in the system, the *Economic* method suffers from serious *request starvation problems*.



(a) Average satisfaction mean

(b) Average satisfaction variability

Fig. 3. Providers' satisfaction in average.

Satisfaction Balance. We now study the *satisfaction balance* (SB) ratios guaranteed by the three methods for various workloads. We assume that providers work out their *satisfaction* as in [7], but without loss of generality, providers can do it differently. For lack of space, we only present the *average satisfaction balance* ($avgSB$) measures, at a discrete time interval $[t_1, t_2]$. The $avgSB$ is defined as the ratio of the smallest and largest *average satisfied* providers, where *average satisfaction* denotes the average of the providers' *satisfaction* at the discrete time interval $[t_1, t_2]$. Furthermore, we measure the *standard deviation* of the providers' *average satisfaction* (*average variability*), in order to evaluate how these methods satisfy all providers. Similarly to the thresholds of *request starvation* and *overutilization*, the *unsatisfaction* threshold over which providers decide to leave, is also quite subjective and may depend on several external factors. Then, to avoid any question on the choice of such a threshold, we assume that providers are not allowed to leave the system by *unsatisfaction*.

Conversely to the RLB results and as expected (because the providers' bids are based on the providers' *intentions*), the $Economic$ method preserves better the providers' *intentions* (see Figure 3(a)). But this occurs because $Economic$ does not pay much attention to the providers' *utilization*. Hence, this method is only suitable for systems where providers do not care about their *utilization* nor response time is very important for consumers. In these results, we have also observed that even if K_nBest ensures the same RLB than $CapacityBased$, it significantly satisfies better providers. Nevertheless, we can observe that both methods have some difficulties to preserve the providers' *intentions* for a very low workload (20% the total system capacity). This is due to the fact that the number of incoming feasible requests is not enough for giving requests to all providers, and thus in both methods, requests are allocated to the least *utilized* providers. Note that K_nBest can be improved in two ways: 1) setting the k_n to a greater value, or 2) in selecting first the k_n most interested providers and allocating requests to the $q.n$ least utilized ones in K_n.

Figure 3(b) shows that, for workloads under 20% of the total system capacity, some providers get more satisfied than others in the *CapacityBased* and K_nBest methods. This is why the *average variability* of both methods is high. However, for higher workloads, the three methods yield almost the same *average variability*.

5 Related Work

In the context of large-scale distributed information systems, most of the work on request allocation [6, 14, 17, 20, 21] has dealt with the problem of allocating requests to the most capable providers. However, preserving providers' *autonomy* for performing requests has not received much attention. Economical methods [3, 4, 16, 13] strive to deal with such *autonomy*. In such models, every provider tries to maximize its revenue by selling services to consumers. Mariposa [13] pioneered the use of an economical method for dealing with the request allocation problem in distributed systems. In Mariposa, all the incoming requests are processed by a *broker site*, which given an incoming request, requests providers for bids. Providers bid for acquiring the request based on a local bulletin board. Then, the *broker site* selects a set of bids that has an aggregate price and delay under a bid curve provided by the consumer. Nevertheless, it is unclear if this method ensures good response times in adequacy with the providers' *intentions*. Furthermore, some requests may not get processed although it exist providers able to deal with it. Recently we have proposed the S_bQLB approach [7] to balance requests across providers while considering their *intentions* for performing requests. S_bQLB strives to balance requests in the course of time thereby reducing *request starvation* in the system. However, K_nBest is orthogonal and more general than all these methods. The K_nBest method is actually a set of solutions for different environments and types of applications.

6 Conclusion

In this paper, we addressed the problem of request allocation in large-scale distributed information systems with autonomous providers. We proposed the K_nBest method which allocates requests by considering providers' *intentions* for performing requests in addition to their *utilization*. The principle of K_nBest is similar to the *two random choices* paradigm which has proven useful for dynamically assigning tasks to providers. We described the implementation of K_nBest in different scenarios with the different strategies to adopt. We gave an experimental evaluation which compares K_nBest to traditional request allocation methods. We demonstrated through experimentation that *CapacityBased* and *Economic* are not suitable for distributed information systems where providers may leave by *request starvation*, *overutilization*, or *unsatisfaction*. The experimental results show that, with autonomous providers in the system, our method significantly outperforms these methods. K_nBest is suitable for dynamic, very

large-scale distributed information systems with competitive or cooperative behaviors. Our method is general and simple, so that it can be easily incorporated in existing distributed information systems. It generalizes traditional methods and can be adapted to the application by varying several parameters.

References

1. O. Morgenstern and J. von Neumann: *Theory of Games and Economic Behavior*. Pinceton University Press, Inc. 1980.
2. A. Sah, J. Blow, and B. Dennis: An introduction to the Rush language. In Procs. of the TCL Workshop. 1994.
3. D. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini: Economic Models for Allocating Resources in Computer Systems. *Market-based control: a paradigm for distributed resource allocation*. World Scientific Publishing Co., Inc. 1996.
4. D. Ferguson, Y. Yemini, and C. Nikolaou: Microeconomic Algorithms for Load Balancing in Distributed Computer Systems. In Procs. of the ICDCS Conference. 1988.
5. E. P. Markatos: Tracing a Large-Scale Peer to Peer System: An Hour in the Life of Gnutella. In CCGRID Symposium. 2002.
6. H. Zhu, T. Yang, Q. Zheng, D. Watson, O. Ibarra, and T. Smith: Adaptive Load Sharing for Clustered Digital Library Servers. In HPDC Symposium. 1998.
7. J.-A. Quiané-Ruiz, P. Lamarre, and P. Valduriez: Satisfaction-Based Query Load Balancing. In Procs. of the CoopIS Conference. 2006.
8. K. Sycara, M. Klusch, S. Widoff, and J. Lu: Dynamic Service Matchmaking Among Agents in Open Information Environments. In SIGMOD Record 28(1). 1999.
9. L. Li and I. Horrocks: A Software Framework for Matchmaking Based on Semantic Web Technology. In Procs. of the WWW Conference. 2003.
10. M. Mitzenmacher: The Power of Two Choices in Randomized Load Balancing. PhD. Thesis, UC Berkeley, 1996
11. M. Nodine, W. Bohrer, and A. Ngu: Semantic Brokering over Dynamic Heterogeneous Data Sources in InfoSleuth. In Procs. of the ICDE Conference. 1999.
12. M. Roussopoulos and M. Baker: Practical Load Balancing for Content Requests in Peer-to-Peer Networks. Distributed Computing 18(6):421-434. 2006.
13. M. Stonebraker, P. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu: Mariposa: A Wide-Area Distributed Database System. In VLDB J. 5(1). 1996.
14. N. Shivaratri, P. Krueger, and M. Singhal: Load Distributing for Locally Distributed Systems. In IEEE Computer Journal 25(12). 1992
15. P. Lamarre, S. Cazalens, S. Lemp, and P. Valduriez: A Flexible Mediation Process for Large Distributed Information Systems. In Procs. of the CoopIS Conference. 2004.
16. R. Buyya, H. Stockinger, J. Giddy, and D. Abramson: Economic Models for Management of Resources in Grid Computing. In CoRR Journal. 2001.
17. R. Mirchandaney, D. Towsley, and J. Stankovic: Adaptive Load Sharing in Heterogeneous Distributed Systems. In Parallel and Distributed Computing J. 9(4). 1990
18. S. Saroiu, P. Krishna Gummadi, and S. Gribble: A Measurement Study of Peer-to-Peer File Sharing Systems. In Procs. of the MCN Conference. 2002.
19. T. Özsu and P. Valduriez: *Principles of Distributed Database Systems*, (2nd ed.). Prentice-Hall, Inc. 1999.
20. Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal: Balanced Allocations. In SIAM Journal on Computing 29(1). 1999.
21. Z. Genova and K. Christensen: Challenges in URL Switching for Implementing Globally Distributed Web Sites. In Procs. of the ICPP Workshops. 2000.